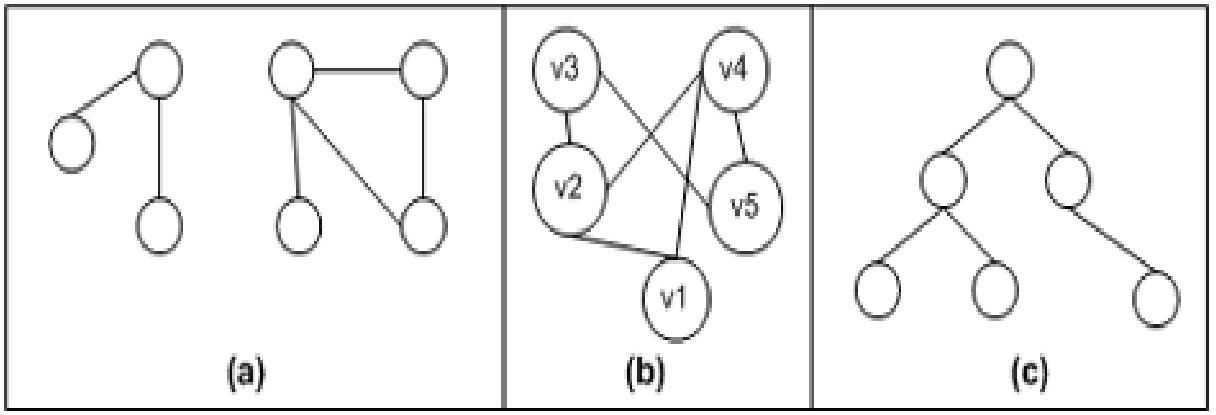
Lecture 9

Graphs

From trees to graphs

- A Tree is a collection of nodes with some restrictions:
  - Starting from the parent node, any other node in the tree can be reached. That is, there exists no node that can't be reached through some simple path.
  - There are no *cycles*. A cycle exists when, starting from some node *v*, there is some path that travels through some set of nodes v_1, v_2, ..., v_k that then arrives back at *v*.
  - The number of edges in a tree is precisely one less than the number of nodes.
- Graphs are composed of a set of nodes and edges, just like trees, but with graphs there are no rules for the connections between nodes. With graphs there is no concept of a root node, nor is there a concept of parents and children. Rather, a graph is just a collection of interconnected nodes.

Graph examples



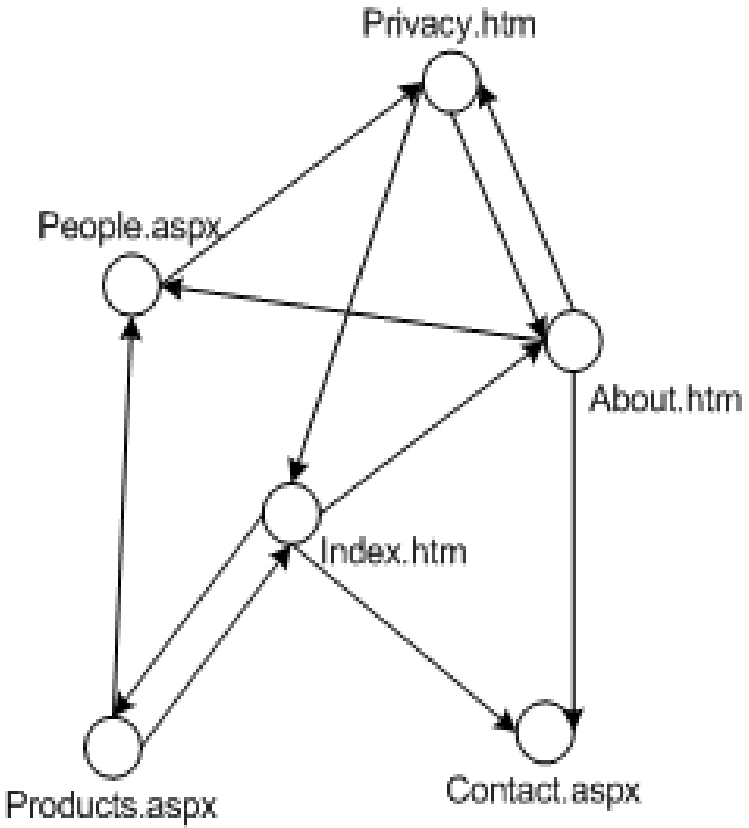
- Unlike trees, graphs can have sets of nodes that are disconnected from other sets of nodes. For example, graph (a) has two distinct, unconnected set of nodes.
- Graphs can also contain cycles. Graph (b) has several cycles
- Graph (c) does not have any cycles, as one less edge than it does number of nodes, and all nodes are reachable. Therefore, it is a tree.

Graphs application

- Graphs are typically used in modeling problem spaces such as:
  - Search engines model the Internet as a graph, where Web pages are the nodes in the graph and the links among Web pages are the edges.
  - Programs that can generate driving directions from one city to another use graphs, modeling cities as nodes in a graph and the roads connecting the cities as edges.

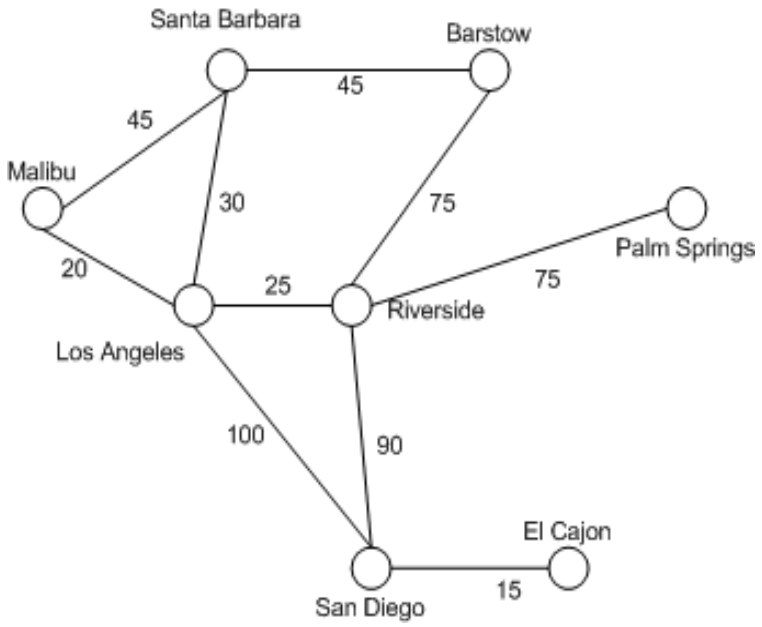
Directed and undirected graphs

- By default, an edge is assumed to be bidirectional. Graphs with bidirectional edges are said to be *undirected graphs*, because there is no implicit direction in their edges.
- Graphs that use unidirectional edges are said to be *directed graphs*. Example: When modeling the internet, a link in page a to page b is a directed edge from the node representing the page a to that representing page b



Weighted and weightless graph edges

- Typically graphs are used to model a collection of "things" and their relationship among these "things.“ Sometimes, it is important to associate some cost with the connection from one node to another.
- A map can be easily modeled as a graph, with the cities as nodes and the roads connecting the cities as edges. If we wanted to determine the shortest distance and route from one city to another, we first need to assign a cost from traveling from one city to another. The logical solution would be to give each edge a *weight*, such as how many miles it is from one city to another.



Graph classification

- Based on the directionality and weighted/weightless edges (two orthogonal criteria), Graphs are classified as:
  - Directed, weighted edges
  - Directed, unweighted edges
  - Undirected, weighted edges
  - Undirected, unweighted edges

Number of edges in a graph

- Directed graphs: Each node in a n nodes graph could be connected to the n-1 other nodes giving n(n-1) edges (maximum number of edges) assuming no self edges (from the node to itself)
- If self edges are possible, the maximum number of edges is n^2.
- If the graph is undirected, then one node, call it v_1, could have an edge to each and every other node, or $n - 1$ edges. The next node, call it v_2, could have at most $n - 2$ edges, because there already exists an edge from v_2 to v_1. The third node, v_3, could have at most $n - 3$ edges, and so forth. Therefore, for n nodes, there would be at most $(n - 1) + (n - 2) + ... + 1$ edges. Summed up this comes to $[n * (n\text{-}1)] / 2$, or, exactly half as many edges as a directed graph.
- If a graph has significantly less than n^2 edges, the graph is said to be *sparse*. For example, a graph with n nodes and n edges, or even $2n$ edges would be said to be sparse. A graph with close to the maximum number of edges is said to be *dense*.

Report Discussion

Last lecture report: Implement the three tree traversal ways (pre-order, in-order and post-order)using iterative loops

New report: How could you model the delta cities map using graphs

Creating a graph class

- If we are going to create a graph class, how can the interconnection between edges and nodes maintained by the class? Two widely used techniques:
  - As an adjacency list
  - As an adjacency matrix

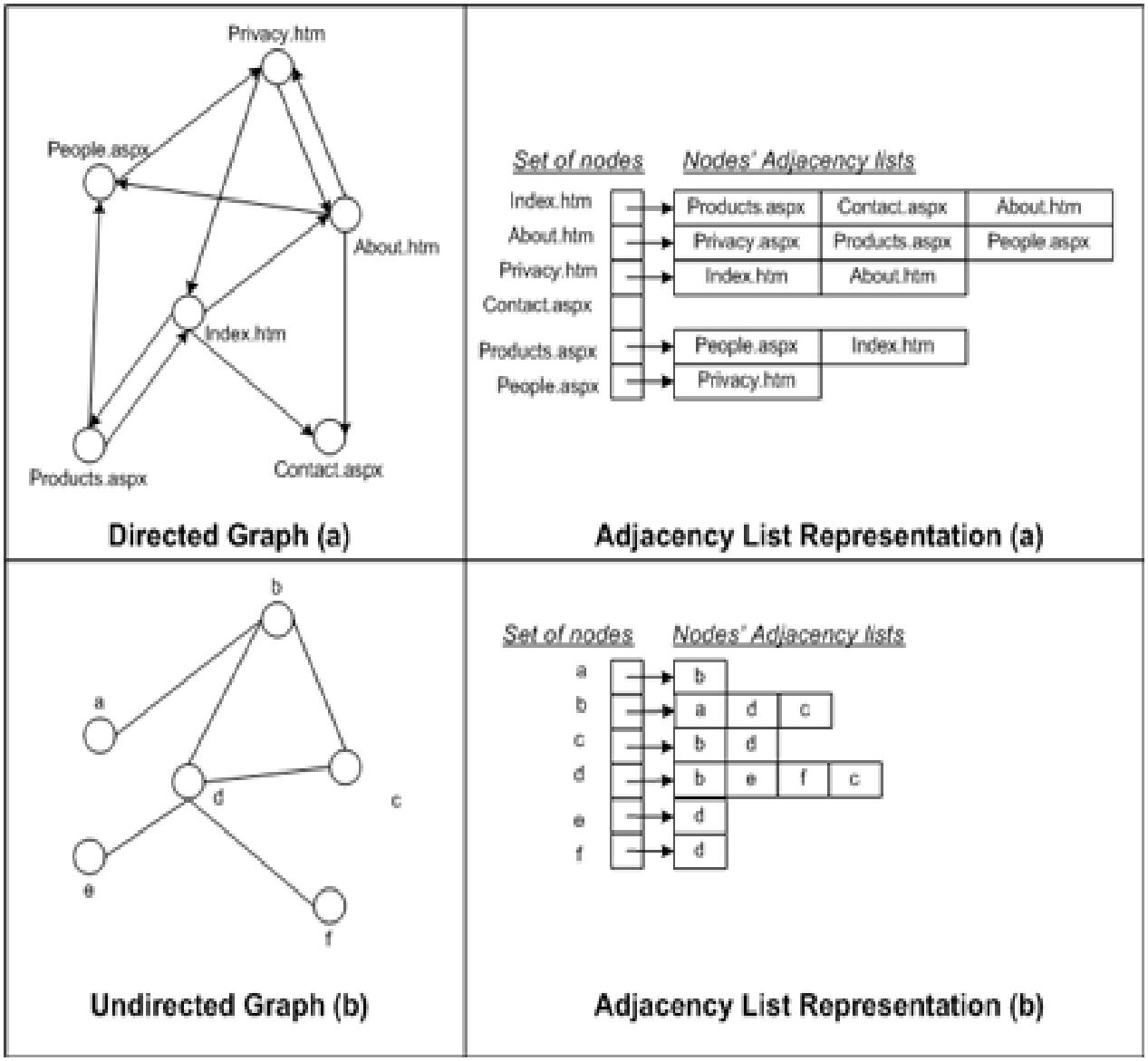
Representing a graph using an adjacency list

- A node represented by a GraphNode class that contains a data and a list of GraphNodes called Neighbors. Of course the graph node class should have a mean to assign cost to each neighbor.
- A Graph class contains a list of GraphNode.
- In this way the Graph class contains a list of GraphNodes and each GraphNode in the list contains a list of its neighbors
- Such a representation is called an *adjacency list*

Adjacency list graph representation examples

With an undirected graph, an adjacency list representation duplicates the edge information. For example, in adjacency list representation (b), the node *a* has *b* in its adjacency list, and node *b* also has node *a* in its adjacency list.

For a graph with *V* nodes and *E* edges, a graph using an adjacency list representation will require *V* + *E* GraphNode instances for a directed graph and *V* + 2*E* *Graph*Node instances for an undirected graph.

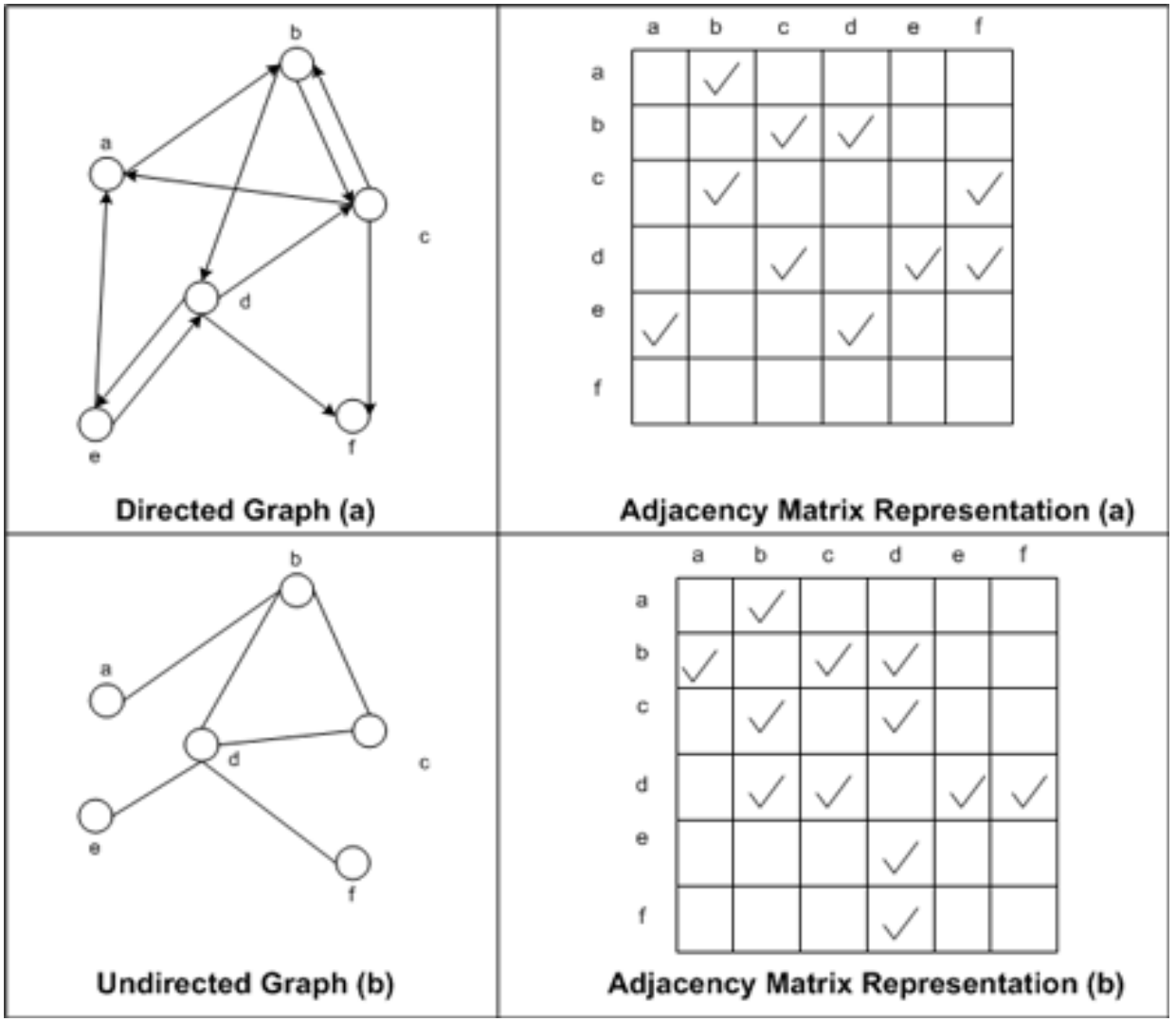


Representing a graph using an adjacency matrix

- For a graph with *n* nodes, an adjacency matrix is an *n* x *n* two-dimensional array. For weighted graphs the array element (*u*, *v*) would give the cost of the edge between *u* and *v* (or, perhaps -1 if no such edge existed between *u* and *v*. For an unweighted graph, the array could be an array of Booleans, where a True at array element (*u*, *v*) denotes an edge from *u* to *v* and a False denotes a lack of an edge.

Representing a graph using an adjacency matrix

Note that undirected graphs display symmetry along the adjacency matrix's diagonal. That is, if there is an edge from *u* to *v* in an undirected graph then there will be two corresponding array entries in the adjacency matrix, (*u*, *v*) and (*v*, *u*).



Comparison between adjacency listand adjacency matrix

Adjacency list

- Space efficient: The adjacency list of a node contains only the nodes with a connection to it
- Determining if a connection exist between two nodes requires linear search time in the Adjacency list of one of them

Adjacency matrix

- Space inefficient, there is a matrix entry for each and every two nodes, even if they are not connected
- Determining if a connection exist between two nodes requires constant search time in the Adjacency matrix